5-2008

Mixtures of Truncated Normal Data Assimilation Models for Parameter Estimation and Prediction in Hydrological Systems

Darl D. Flake II
Utah State University

MIXTURES OF TRUNCATED NORMAL DATA ASSIMILATION MODELS FOR PARAMETER ESTIMATION AND PREDICTION IN HYDROLOGICAL SYSTEMS

by

Darl D. Flake II

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Statistics

UTAH STATE UNIVERSITY
Logan, Utah

2008

Mixtures of Truncated Normal Data Assimilation Models for Parameter Estimation and Prediction in Hydrological Systems

Darl D. Flake II

Abstract

Physical models in the hydrological sciences are often calibrated using methods that do not formally quantify uncertainty in the model parameters. Additionally, many competing hydrological models exist and are used to model the same processes. Considering existing mechanistic models of rainfall-runoff in a statistical context can assist hydrologists in understanding the true physical process taking place. This paper introduces a data assimilation mixture model of runoff that yields statistical estimates of hydrological model parameters and predictions. This statistical model incorporates two commonly used hydrological models, each with strengths and weaknesses. The mixture framework allows comparisons between models as well as combines the strengths of both. Results from three implementations of the mixture model are summarized and additional generalizations of the models are suggested.

1 Introduction

In the hydrological sciences, rainfall-runoff is often studied using conceptual models (Brutsaert, 2005), that represent the physical process that water inputs undergo

as they change from rain to runoff, using a series of mathematical rules. Model parameters are unmeasureable values that control different aspects of these rules (Schaake, 2003). Examples of model parameters are capacities of conceptual underground tanks and rates at which water flows between them. Conventionally, in order to predict runoff, these models require sophisticated calibration techniques to determine appropriate parameter values (e.g., Huard and Mailhot 2006; Muleta and Nicklow 2005). Frequently, such calibration is performed without formally addressing uncertainty inherent in the process model (Liu and Gupta, 2007; Vrugt et al., 2005). Statistical models can be used to rigorously account for model uncertainty, though implementation is non-trivial. Typically, the runoff models are highly non-linear and algorithmic, such that they can be carried out on a computer but not easily written in closed form (Vogel and Sankarasubramanian, 2003). Additionally, the fact that runoff, as a random variable, has non-negative support adds to the complexity of statistical implementations of hydrological models. Another challenge to modeling runoff statistically is that many conceptual models can be used to describe the same phenomenon. In these cases, choosing one over the other is often very subjective; for example, the model that gives the best overall predictions or the more elaborate model could be chosen. It may also be the case that one model performs better than others only in certain circumstances.

Some of these difficulties can be addressed using traditional techniques, but there remains room for improvement. For example, issues involving non-negative support in the response can sometimes be dealt with by means of a transformation but this is not always most appropriate. Traditional statistical methods cannot easily be used to estimate the runoff model parameters due to the significant non-linearity in the process. They also have no mechanism for incorporating prior scientific knowledge about the hydrological process under study.

Many difficulties encountered when modeling runoff from a statistical perspective

can be overcome naturally by using the Bayesian framework for model specification and fitting. Any number of probability models with positive support (e.g., gamma, log-normal, etc.) can be used to model the response variable without increasing the complexity of the analysis method. Complicated non-linear models can also be specified and implemented relatively easily within a Bayesian framework. In such cases, Bayesian models require numerical integration methods. Therefore, because both conceptual runoff models and Bayesian models are often implemented using highly computational algorithms, the formal combination of the two is straightforward.

2 Methods

We propose a Bayesian approach to data assimilation for runoff that directly takes into account its non-negative support by utilizing a reparameterized truncated Gaussian mixture model. This eliminates the need to choose one model over another and instead combines the strengths from each.

As an example, we consider two deterministic models from hydrology: the Hydrologic Model known as HYMOD and the Sacramento Soil Moisture Accounting model (SAC-SMA; Burnash et al. 1973). HYMOD is a relatively simple five parameter model developed at the University of Arizona and used mainly as a teaching tool, whereas SAC-SMA is a much more complicated model with up to 23 parameters (depending on which are assumed to be constant and known) used by the National Weather Service. Note that hydrologists may not consider all these as parameters that need to be estimated statistically.

Using computational Bayesian methods we apply these two models to a widely used, historical data set from the Leaf River Basin in Mississippi, U.S.A. (e.g., Vrugt et al. 2005; Misirli et al. 2003). These data include daily measurements of runoff, potential evapotranspiration (PET), and precipitation. HYMOD and SAC-SMA take

PET and precipitation as inputs. We let $\boldsymbol{x}_t$ represent these inputs for each day and $\boldsymbol{X}_t = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t\}$ represent the PET and precipitation for all days up to and including the day t. Runoff is the output for each of the models and, for a particular day t, is represented by y_t. Our analyses use the data from a six year period from January 1, 1957 to December 31, 1962.

In what follows, we will introduce the framework for an individual model first (Section 2.1), then follow with the extension to the mixture model setting (Section 2.2).

2.1 Bayesian Data Assimilation

First we will describe the statistical model we use to consider the physical process models HYMOD and SAC-SMA in a stochastic data assimilation setting (Wikle and Berliner, 2007). The model we propose can be described hierarchically in terms of three components: the data model, the process model, and the parameter model (Berliner, 1996). We will begin by describing the data model.

2.1.1 Data Model

Let y_t represent the observed runoff at time t, for $t = 1, \ldots, T$. We assume these observations arise from HYMOD or SAC-SMA, using the following probability model:

$$y_t \sim \mathrm{TN}\left(g\left(f_t, \sigma_y^2\right), \sigma_y^2\right)_0^\infty, \tag{1}$$

where output from HYMOD or SAC-SMA at time t is represented by f_t, and g is a reparameterization of the mean described below. The 0 and the ∞ in (1) are, respectively, the lower and upper bounds of the truncation of a Normal distribution. Note that infinite runoff is not necessarily realistic and a finite upper bound could also be specified if physical properties of the system indicate one.

Now, recall that a truncated normal distribution is a normal distribution that has zero probability below the lower truncation and zero probability above the upper truncation. The remaining density is scaled such that it integrates to one. The distribution's parameters are the expected value and variance of the non-truncated distribution (Johnson et al., 1994). A feature critical to our model specification is that the expected value of a normal distribution that has been truncated on the left will be greater than the expected value of the non-truncated distribution. In this case, to specify a sensible probability model we let the expected value of y_t equal f_t. Although it is natural to specify the mean of the non-truncated normal distribution, it requires a reparameterization to specify the mean of the truncated normal distribution. In the case of left truncation at zero, as specified in (1), the expected value of a truncated normal distribution can be written in terms of the mean and variance of the non-truncated distribution:

$$\mathrm{E}(y_t) = g + \sigma_y \left(\frac{\phi\left(\frac{g}{\sigma_y}\right)}{\Phi\left(\frac{g}{\sigma_y}\right)} \right), \tag{2}$$

where ϕ is the probability density function and Φ is the cumulative density function of a standard Normal distribution (Johnson et al., 1994; Jawitz, 2004). Letting the $\mathrm{E}(y_t)$ in (2) equal f_t and then inverting the equation yields the desired mean parameter that results in a truncated normal distribution with expected value equal to f_t. This inverse function, which we call g, is a function of the desired expected value, f_t, and the variance of the non-truncated distribution, σ_y^2. Although g exists, it is analytically intractible and must be found numerically (See Appendix A for details). In summary, the function $g\left(f_t, \sigma_y^2\right)$ in (1), serves as a model reparameterization and ensures that $\mathrm{E}(y_t) = f_t$.

As with the mean parameter, the variance parameter (σ_y^2) is the variance of the non-truncated normal distribution. It accounts for the stochasticity in the model beyond that provided by the parameters $(\boldsymbol{\theta})$, which are in turn allowed to be random.

Note that, the variance of y_t is less than σ_y^2 due to our reparameterization, however the variance of y_t approaches σ_y^2 as f_t approaches infinity. In this case, the reduced variability near the truncation bound is a desirable quality because the process under study exhibits the same behavior.

2.1.2 Process Model

In the following specification, the process model contains the conceptual runoff model of interest. It is denoted by f_t, a function of the input data, $\boldsymbol{X}_t$ (i.e. PET and precipitation), and the $p \times 1$ vector of model specific parameters, $\boldsymbol{\theta}$. Here, the process model is a hydrological model, with no closed form. In general, the process could be written as:

$$f|\boldsymbol{\theta} \sim [f|\boldsymbol{\theta}], \tag{3}$$

where the stochastic form of $[f|\boldsymbol{\theta}]$ could be specified in terms of additional parameters. Note that, here and throughout the rest of this paper, the square brackets represent the probability density function of the enclosed variables (e.g. $[f|\boldsymbol{\theta}]$ is the pdf of the random variable $f|\boldsymbol{\theta}$). In this case, we assume f is only stochastic through the hydrological parameters $\boldsymbol{\theta}$, thus $f_t = f(\boldsymbol{X}_t, \boldsymbol{\theta})$.

2.1.3 Parameter Model

The parameters of the statistical model are the parameters controlling the hydrological proccess ($\boldsymbol{\theta}$) and the runoff variance (σ_y^2). The hydrological model parameters are conventionally thought to have bounded support, so we model them as follows:

$$\boldsymbol{\theta} \sim \mathrm{TN}(\boldsymbol{\mu}_\theta, \Sigma_\theta)_{\boldsymbol{a}_1}^{\boldsymbol{a}_2}, \tag{4}$$

where Σ_θ is a diagonal matrix with entries that correspond to the prior variance of each of the parameters. The vectors $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are the bounds on the parameter

support.

The variance components are then modeled as:

$$\log\left(\sigma_y^2\right) \sim \mathrm{N}(\lambda, \tau^2). \tag{5}$$

Note that a model for variance (σ_y^2) could be specified in many other ways, here we found the log-normal model convenient in terms of assigning vague prior knowledge about the model uncertainty. Additionally, the eventual implementation is no more complex than for other common variance models (e.g., the inverse-gamma model), due to the non-conjugacy created by the data model.

2.2 Mixture Model

By modeling runoff as a mixture of process models, we can evaluate the effectiveness of each as well as improve predictions. This mixture model specification can also be written in terms of data, process, and parameter models.

2.2.1 Data Model

Consider observed runoff as arising from a mixture of two stochastic models as follows:

$$y_t \sim \begin{cases} \mathrm{TN}\left(g\left(f_{t1}, \sigma_1^2\right), \sigma_1^2\right)_0^\infty, & \text{w.p. } \psi_t, \\ \mathrm{TN}\left(g\left(f_{t2}, \sigma_2^2\right), \sigma_2^2\right)_0^\infty, & \text{w.p. } 1-\psi_t, \end{cases} \tag{6}$$

where ψ_t is the mixture probability or the probability that y_t arises from HYMOD (f_{t1}) instead of SAC-SMA (f_{t2}) at time t. The specification in (6) yields the following likelihood:

$$\left[\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \sigma_1^2, \sigma_2^2\right] \propto \prod_{t=1}^{T} \left[y_t|\boldsymbol{\theta}_1, \sigma_1^2\right]^{z_t} \left[y_t|\boldsymbol{\theta}_2, \sigma_1^2\right]^{1-z_t}, \tag{7}$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_t, \ldots, y_T)'$, $\boldsymbol{z} = (z_1, z_2, \ldots, z_t, \ldots, z_T)'$, and z_t (defined below) is an auxilliary, indicator variable that controls the mixing.

2.2.2 Process Model

The process model for the mixture does not change from the process model for the individual data assimilation models. However, there are two such models in the mixture. Particularly,

$$f_{t1} = f_1(\boldsymbol{X}_t, \boldsymbol{\theta}_1), \tag{8}$$

represents HYMOD and

$$f_{t2} = f_2(\boldsymbol{X}_t, \boldsymbol{\theta}_2), \tag{9}$$

represents SAC-SMA. Once again, f_1 and f_2 are not distributions in this specific case. The only uncertainty considered in the process model is that accounted for in the hydrological models' parameters $\boldsymbol{\theta_1}$ and $\boldsymbol{\theta_2}$, although the model could be extended to more general forms.

2.2.3 Parameter Model

The two sets of parameters of the hydrological models and the variance components are assumed to be independent and are modeled in the same fashion as previously described. We must now consider the parameters involved in the mixing. In this case, it is helpful to introduce the auxilliary variables z_t as follows:

$$z_t|\psi_t \sim \text{Bernoulli}(\psi_t), \tag{10}$$

where, the mixture probabilities ψ_t are then modeled as:

$$\psi_t \sim \text{Beta}(\alpha, \beta). \tag{11}$$

Now, ψ_t have the same interpretation as in (6) but the introduction of z_t facilitates model implementation. Implementation is also simplified by the fact that we chose to model the ψ_t independently even though model preference could be correlated through time. Lack of a specific form for this possible correlation is a compelling reason to use the exchangeable prior in (11).

2.3 Implementation

We implemented the mixture in three ways. First, there is the general case where ψ_t and z_t are different for each time step t (i.e. the "ψ_t, z_t" model). The "ψ_t, z_t" model can be used to compare HYMOD and SAC-SMA at individual time points t. However, it cannot be used to predict runoff at unknown time points. If one desires to use the mixture for prediction while still allowing the individual hydrological models to describe runoff at each time point then the "ψ, z_t" model is the implementation of choice. As indicated in its name, there is only one mixture probability in this model (i.e. $\psi_t = \psi, \forall t$). The last model we implemented, the "ψ, z" model, occurs when ψ_t is constrained to equal ψ and z_t to equal z for all t (i.e. $\psi_t = \psi, z_t = z, \forall t$). This would specify a model that is mixed at all time points simultaneously. The "ψ, z" model can be used to decide which hydrological model is a more probable global representation of runoff.

We are implementing these models from a Bayesian perspective, thus we seek to learn about the parameters using the posterior distribution (i.e. the joint distribution of the parameters given that data have been observed):

$$\left[\psi, \boldsymbol{z}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \sigma_1^2, \sigma_2^2 | \boldsymbol{y}\right] \propto \prod_{t=1}^{T} \left(\left[y_t | \boldsymbol{\theta}_1, \sigma_1^2\right]^{z_t} \left[y_t | \boldsymbol{\theta}_2, \sigma_1^2\right]^{1-z_t} \left[z_t | \psi_t\right] \left[\psi_t\right] \right) \left[\boldsymbol{\theta}_1\right] \left[\boldsymbol{\theta}_2\right] \left[\sigma_1^2\right] \left[\sigma_2^2\right]. \tag{12}$$

Marginal posterior distributions for all of the parameters for each statistical model are calculated by integrating out the rest of the parameters from the joint posterior

distributions. For both data assimilation models and all three implementations of the mixture model we calculated the posterior predictive distribution as well:

$$[y_t|\boldsymbol{y}] = \int \cdots \int [y_t|\boldsymbol{\psi}, \boldsymbol{z}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \sigma_1^2, \sigma_2^2] \left[\boldsymbol{\psi}, \boldsymbol{z}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \sigma_1^2, \sigma_2^2|\boldsymbol{y}\right] d\boldsymbol{\psi} d\boldsymbol{z} d\boldsymbol{\theta}_1 d\boldsymbol{\theta}_2 d\sigma_1^2 d\sigma_2^2 \tag{13}$$

These posterior distributions are highly non-conjugate and analytically intractable and thus a Markov Chain Monte Carlo (MCMC) approach with Metropolis updates was taken to fit the proposed models. (See Appendix B for details).

After fitting the models and fitting the posterior and posterior predictive distributions we calculated the Deviance Information Criteria (DIC) for each model. The DIC is used as a measure of goodness of fit and serves as a means to compare the various models (i.e., the individual data assimilation models as well as the three implementations of the mixture model).

3 Results

First we implemented the individual data assimilation models for HYMOD and SAC-SMA. Bounds for the truncated normal distributions can be found in Table 1 for HYMOD and Table 2 for SAC-SMA. We set $\boldsymbol{\mu}_\theta$ equal to the midpoint of the truncation bounds and $\boldsymbol{\Sigma}_\theta$ is a diagonal matrix with entries equal to $100\boldsymbol{\mu}_\theta$. The Normal distribution on $\log(\sigma_y^2)$ was specified with a mean (λ) of 9.3 and a variance (τ^2) of 10,000, which yields a vague prior.

The mixture models used these same hyperparameter values for the hydrological models and regardless of the particular implementation of the mixture model, the prior beta distributions on ψ_t were assigned α and β hyper-parameters equal to 1, indicating no preference for which model is more likely, *a priori*.

All models were fit using an MCMC algorithm that was run for 10,000 iterations. After a burn-in of 1,000 iterations, chains were thinned and the retained samples were

used to produce the distributions of interest.

The posterior predictive distribution for HYMOD and SAC-SMA can be found in Figures 1 and 2. Tables 1 and 2 compare 95% credible intervals for the posterior distributions and the prior distributions for all of the parameters in HYMOD and SAC-SMA, respectively. Posterior distributions of σ_1^2 and σ_2^2 can be found in Figure 8.

The posterior predictive distributions from the "ψ_t, z_t" and the "ψ, z_t" models are presented in Figures 3 and 4, respectively. All samples simulated from the posterior distribution of z, in the "ψ, z" model were equal to zero. In other words, this mixture resulted in the same posterior predictive distribution as the SAC-SMA model. Mixing for the "ψ_t, z_t" model is summarized in the following way: Figure 5 displays boxplots of the posterior distribution of the mixture probability at each time point ($[\psi_t|\boldsymbol{y}]$) and asymptotic confidence intervals for the expectation of z_t (calculated from the MCMC samples using frequency based methods) for each time step can be found in Figure 6. The posterior distribution of ψ, for the "ψ, z_t" model can be found in Figure 9. The asymptotic confidence intervals for the expectation of z_t is in Figure 7. The DIC for all of the models can be found in Table 3.

4 Discussion

One result of modeling runoff as a truncated normal distribution is the effect it has on the variance of the runoff values. If a normal distribution is truncated on the left, the resulting expectation will be greater than that of the original distribution. This is the reason the function g was utilized to reparameterize the truncated normal distribution in (1), so that its mean is constrained to be f_t (i.e. the hydrological model output). Likewise, when a normal distribution is truncated, the variance of the resulting distribution is smaller than the variance of the non-truncated distribution

(Johnson et al., 1994; Jawitz, 2004). In our models, the truncation is at zero, thus for small values of runoff the variance is reduced. This implies that, even though the variance parameter is specified to be constant, truncation causes the variance to be dependent on the mean parameter and therefore different for each time step. This is evident in the plots of the posterior predictive distributions (Figures 1 through 4). The runoff at each time point comes from a truncated normal distribution with the same variance parameter (σ_y^2) and different mean parameters ($g\left(f_t, \sigma_y^2\right)$), but there is less uncertainty about smaller runoff values; this is an important result of the truncation.

Other benefits that come from the truncated normal model of runoff are its flexibility and ability to accommodate zeros in observed values. Although there are many distributions with positive support that could be used to model runoff, all common distributions, including the two mentioned in the introduction (gamma and log-normal), do not include zero in their support. A truncated normal distribution has non-zero probability density at its truncation points (Johnson et al., 1994). Our model is truncated at zero, which allows runoff in-turn to be zero. The truncated normal also has a very flexible stochastic form for modeling runoff. When runoff values are large and therefore far from the trucation point, the truncated distribution is Gaussian in shape. In contrast, when runoff values are near zero the probability distribution takes on a skewed form, more characteristic of a gamma distribution.

HYMOD and SAC-SMA can be compared on the scale of the entire time series in two ways. The most straight forward is the model DIC. The SAC-SMA model has a lower DIC than HYMOD. This indicates that, overall, SAC-SMA does better at describing the true process. The other method is to compare HYMOD and SAC-SMA using the results from the "ψ, z" model. Out of a large number of MCMC samples HYMOD was never preferred in the mixing of the two models. This implies that, for this dataset SAC-SMA is an overwhelmingly more probable model than HYMOD.

Additional evidence of SAC-SMA outperforming HYMOD is that the posterior distribution of the variance component ($[\sigma_y^2|\boldsymbol{y}]$), illustrated in Figure 8, is smaller for Sacramento than it is for HYMOD. This results in the discrepancy in uncertainty regarding runoff predictions each of the models (Figures 1 and 2). One interpretation is that the variance component for HYMOD has to be larger to make up for inadequacies in the physical model.

The two stochastic models can also be compared at each time point. By looking at the posterior predictive distributions of the separate models (Figures 1 and 2) it is evident that SAC-SMA and HYMOD predict the true runoff values better in some places than in others. For example, around time point $t = 1515$, SAC-SMA is able to capture the high runoff better than HYMOD. However, around time point $t = 1630$, HYMOD does better at predicting the lower runoff values than SAC-SMA. The boxplots of $[\psi_t|\boldsymbol{y}]$ from the "ψ_t, z_t" model show at which time points HYMOD performs better. Note that, due to the overparameterization in the "ψ_t, z_t" model the most extreme distribution ψ_t can have is a Beta $(\alpha + z_t, \beta + (1 - z_t))$. Therefore, to be able to carry out a formal test to see if one model is significantly more probable than another at a particular time point we consider the expectation of z_t. Using the MCMC samples to approximate the expectation yields a proportion statistic which is asymptotically normal. Ninety-five percent confidence intervals for the expectation of z_t in Figure 6 serve as hypothesis tests at α-level 0.05 that the observed proportion is significantly different from 0.5. So, in the area around $t = 1515$, SAC-SMA is significantly more probable than HYMOD, and in the area around $t = 1630$, HYMOD is significantly more probable than SAC-SMA; whereas from time points $t = 1580$ to $t = 1625$ neither model is preferred over the other.

Although the "ψ_t, z_t" model has the lowest DIC of the mixture models and it allows comparison of models at specific times, it cannot be used for prediction. This is due to a lack of knowledge about future mixture probabilities (ψ_t). The "ψ, z"

model can be used to predict unobserved runoff, but it is not flexible enough to allow the weaker HYMOD model to be utilized. The "ψ, z_t" model is a compromise between the two. (i.e., its DIC is between those of the "ψ, z" and "ψ_t, z_t" models in Table 3). The "ψ, z_t" model allows mixing at each time point and thus, ψ can be used for future predictions. In the "ψ, z_t" model, the mode of the posterior distribution on the mixture probability (i.e., ψ, Figure 9) was around 0.15, which shows that even though HYMOD is allowed to contribute to the model at every time point it only does so about 15% of the time. At individual time points HYMOD can still be significantly more probable than SAC-SMA as measured by the expectation of z_t. Figure 7 provides an illustration of this behavior at time point $t = 1551$.

5 Conclusion

We have illustrated that runoff can be modeled stochastically using a mixture of existing hydrological models. Using a Bayesian approach we account for non-negative support in the response as well as compare and combine the strengths of distinct conceptual models. Considering the hydrological models in a statistical framework, via data assimilation, accounts for the uncertainty in the model parameters and allows for prediction of non-negative runoff values. The mixture aspect of the model facilitates comparision of the models as well as improves prediction.

Some of the benefits of using a truncated normal distribution to model runoff have been discussed. There are many extensions of this model that can be explored. One particular area of ongoing research is in the development of spatial models (i.e., distributed models in the hydrology literature) based on the univariate model proposed here. Such models can be specified with relative ease due to the fact a truncated normal distribution can be generalized to the correlated multivariate case in the same manner as a non-truncated normal distribution. It should also be noted that this

statistical approach can be used with many different hydrological models, not just HYMOD and SAC-SMA, and more than two models can be considered in a mixture. Another area of potential for the mixture model, specifically the "ψ_t, z_t" model, is that it could be extended to allow for autoregressive modeling of the mixture probability. This would increase the predictive ability of these mixture models. Additional model structure could also be used for separating measurment error and process uncertainty as well as account for uncertainty in the model inputs. Extensions of the model could also accommodate multiple inputs and outputs that could be measured on different scales. More generally, the methods we employed to model runoff as a truncated normal distribution can also be used to model other physical processes that have values with non-negative support (e.g., snowpack dynamics, population biology; Cangelosi and Hooten 2008).

References

Berliner, L. (1996). Hierarchical Bayesian time series models. In K. Hanson and R. Silver (Eds.), *Maximum Entropy and Bayesian Methods*, pp. 15–22. Kluwer Academic Publishers.

Brutsaert, W. (2005). *Hydrology: An Introduction.* New York: Cambridge University Press.

Burnash, R., R. Ferral, and R. McGuire (1973). A generalized streamflow simulation system: conceptual models for digital computers. Technical report, Joint Federal-State River Forecast Center, Sacramento, CA.

Cangelosi, A. and M. Hooten (2008). Models for bounded systems with continuous dynamics. *Biometrics*. In Review.

Huard, D. and A. Mailhot (2006). A Bayesian perspective on input uncertainty in model calibration: Application to hydrological model "abc". *Water Resources Research 42*, W07416.

Jawitz, J. (2004). Moments of truncated continuous univariate distributions. *Advances in Water Resources 27*, 269–281.

Johnson, N., S. Kotz, and N. Balakrishnan (1994). *Continuous Univariate Distribuions.* New York, NY: John and Wiley and Sons.

Liu, Y. and H. Gupta (2007). Uncertainty in hydrologic modeling: Toward an integrated data assimilation framework. *Water Resources Research 43*, W07401.

Misirli, F., H. Gupta, S. Sorooshian, and M. Thiemann (2003). Bayesian recursive estimation of parameter and output uncertainty for watershed models. In Q. Duan, H. Gupta, S. Sorooshian, A. Rousseau, and R. Turcotte (Eds.), *Calibration of Watershed Models*, pp. 113–124. Washington, DC: American Geophysical Union.

Muleta, M. and J. Nicklow (2005). Sensitivity and uncertainty analysis coupled with automatic calibration for a distributed watershed model. *Journal of Hydrology 306*, 127–145.

Schaake, J. (2003). Introduction. In Q. Duan, H. Gupta, S. Sorooshian, A. Rousseau, and R. Turcotte (Eds.), *Calibration of Watershed Models*, pp. 1–8. Washington, DC: American Geophysical Union.

Vogel, R. and A. Sankarasubramanian (2003). Validation of a watershed model without calibration. *Water Resources Research 39*, 1292.

Vrugt, J., C. Diks, H. Gupta, W. Bouten, and J. Verstraten (2005). Improved treatment of uncertainty in hydrologic modeling: Combining the strengths of global optimization and data assimilation. *Water Resources Research 41*, W01017.

Wikle, C. and L. Berliner (2007). A Bayesian tutorial for data assimilation. *Physica D. 230*, 1–16.

Appendix A: Finding the function $g(f_t, \sigma_y^2)$

A simple root finding method for approximating g that works well in matrix languages is linear interpolation. The basic idea is as follows, we can calculated $\mathrm{E}(y_t)$ for many values of g and choose the two values that are closest to the desired expected value. Then the line that connects these two points is an approximation to the true function and the g that is necessary to obtain this desired expected value can be found. This method is slow if one does not know where to choose the test points for g. Ideally, one would choose test g values where the function is changing the most since the linear approximation will be accurate where $\mathrm{E}(y_t)$ is approximately linear. An illustration of $\mathrm{E}(y_t)$ can be found in Figure 10, where the variance (σ_y^2) for the non-truncated normal is equal to 4000 although the general shape is the same for all variances. As g approaches negative infinity the slope of $\mathrm{E}(y_t)$ approaches zero; and as g approaches positive infinity the slope of $\mathrm{E}(y_t)$ approaches one. Therefore, the derivative of $\mathrm{E}(y_t)$ is a cumulative distribution function. If we could generate test g values from this distribution our approximations would have high precision. However, in such a case, we would also have a closed form for g. Instead, we let the test values for g be equally spaced quantiles of a $\mathrm{N}\left(0, \sigma_y^2\right)$ distribution, which approximates the desired distribution. This results in a higher density of points in places where $\mathrm{E}(y_t)$ is changing the most. Note that numerous other methods exist for finding roots of equations and some may outperform others in various circumstances.

Appendix B: Markov Chain Monte Carlo Algorithm

MCMC methods require sampling from the full-conditional distributions of each of the parameters. These full-conditional distributions are:

$$
\begin{aligned}
[\boldsymbol{\theta}_j|\cdot] &\propto \prod_{t\in T_j} \mathrm{TN}\left(g\left(f_t,\sigma_y^2\right),\sigma_y^2\right)_0^\infty \times \mathrm{TN}\left(\boldsymbol{\mu}_\theta,\boldsymbol{\Sigma}_\theta\right)_{a_1}^{a_2} \\
[\sigma_j^2|\cdot] &\propto \prod_{t\in T_j} \mathrm{TN}\left(g\left(f_t,\sigma_y^2\right),\sigma_y^2\right)_0^\infty \times \mathrm{LN}(\lambda,\tau^2) \\
[z_t|\cdot] &\sim \mathrm{Bernoulli}\left(\frac{\psi_t\left[y_t|\boldsymbol{\theta}_1,\sigma_1^2\right]}{\psi_t\left[y_t|\boldsymbol{\theta}_1,\sigma_1^2\right]+(1-\psi_t)\left[y_t|\boldsymbol{\theta}_2,\sigma_2^2\right]}\right) \\
[\psi_t|\cdot] &\sim \mathrm{Beta}\left(\alpha+z_t,\beta+(1-z_t)\right),
\end{aligned}
$$

where $T=\{T_1,T_2\}'$. and j is the number of models in the mixture.

Given initial values $\boldsymbol{\theta}_1^{(0)}$, $\boldsymbol{\theta}_2^{(0)}$, and $z_t^{(0)}$ the following steps are repeated until sufficient samples have been obtained after convergence:

1. Sample $\sigma_j^{2(k)}$, for all j.

   (a) Sample $\sigma_j^{2(*)}$ from a truncated normal proposal distribution.

   (b) Calculate the Metropolis ratio:

$$
\mathrm{R}=\frac{\prod_{t\in T}\left[y_t|\boldsymbol{\theta}_j^{(*)},\sigma_j^{2(k-1)}\right]\left[\sigma_j^{2(*)}\right]\left[\sigma_j^{2(k-1)}|\sigma_j^{2(*)}\right]}{\prod_{t\in T}\left[y_t|\boldsymbol{\theta}_j^{(k-1)},\sigma_j^{2(k-1)}\right]\left[\sigma_j^{2(k-1)}\right]\left[\sigma_j^{2(*)}|\sigma_j^{2(k-1)}\right]}
$$

   (c) Let $\sigma_j^{2(k)}=\sigma_j^{2(*)}$ with probability $\min(\mathrm{R},1)$.

2. Sample $\boldsymbol{\theta}_j{}^{(k)}$, for all j.

   (a) Sample $\boldsymbol{\theta}_j^{(*)}$ from a truncated normal proposal distribution.

(b) Calculate the Metropolis ratio:

$$\mathrm{R} = \frac{\prod_{t \in T} \left[y_t | \boldsymbol{\theta}_j^{(*)}, \sigma_j^{2(k)} \right] \left[\boldsymbol{\theta}_j^{(*)} \right] \left[\boldsymbol{\theta}_j^{(k-1)} | \boldsymbol{\theta}_j^{(*)} \right]}{\prod_{t \in T} \left[y_t |, \boldsymbol{\theta}_j^{(k-1)}, \sigma_j^{2(k)} \right] \left[\boldsymbol{\theta}_j^{(k-1)} \right] \left[\boldsymbol{\theta}_j^{(*)} | \boldsymbol{\theta}_j^{(k-1)} \right]}$$

(c) Let $\boldsymbol{\theta}_j{}^{(k)} = \boldsymbol{\theta}_j{}^{(*)}$ with probability $\min(\mathrm{R}, 1)$.

3. Sample $\psi_t^{(k)}$ from $\left[\psi_t | y_t, \boldsymbol{\theta}_1{}^{(k)}, \boldsymbol{\theta}_2{}^{(k)}, \sigma_1^{2(k)}, \sigma_2^{2(k)}, z_t^{(k-1)} \right]$, for all t.

4. Sample $z_t^{(k)}$ from $\left[z_t | y_t, \boldsymbol{\theta}_1{}^{(k)}, \boldsymbol{\theta}_2{}^{(k)}, \sigma_1^{2(k)}, \sigma_2^{2(k)}, \psi_t^{(k)} \right]$, for all t.

Appendix C: R Code

HYMOD

Following is the code used to run the MCMC routine for HYMOD.

```
'simpRRDAmcmc' <-
function (y, x, s2mn, s2var, ngibbs, outdat = 10)
{
###
###  Subroutines
###
    require(msm)
    dyn.load("HyMod.so")
    hymod.C <- function(pet, precip, parameters) {
        .C("HyMod", as.double(pet), as.double(precip), as.double(parameters),
            as.integer(length(pet)), runoff = as.double(rep(0,
                length(pet))))$runoff
    }
    dinvgamma <- function(x, q, r) {
        1/(r^q * gamma(q)) * (x^(-(q + 1))) * exp(-1/(r * x))
    }
    logdinvgamma <- function(x, q, r) {
        -q * log(r) - lgamma(q) - (q + 1) * log(x) - 1/(r * x)
    }
    rtn <- function(mu, sig) {
        mu + sig * qnorm(log(runif(length(mu))) + pnorm(mu/sig,
            log = TRUE), lower.tail = FALSE, log.p = TRUE)
    }
    logdtnorm <- function(x, mean = 0, sd = 1) {
        lower = 0
```

```
        upper = Inf
        ret <- numeric(length(x))
        ret[x < lower | x > upper] <- 0
        ind <- x >= lower & x <= upper
        if (any(ind)) {
            xtmp <- dnorm(x, mean, sd, log = TRUE) - pnorm(mean/sd,
                log = TRUE)
            ret[x >= lower & x <= upper] <- xtmp[ind]
        }
        ret
    }
    getgvec <- function(f, sig, n) {
        l = -100
        N = length(f)
        u = seq(1e-11, 0.999999999999, , n)
        gtmp = sig * qnorm(u, , sig)
        n = n + 1
        gtmp = c(gtmp, max(f))
        G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) - pnorm(gtmp/sig,
            log = TRUE)))
        l = min(gtmp)
        while (min(G) >= min(f)) {
            l = l - 100
            gtmp = c(l, gtmp)
            n = n + 1
            G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) -
                pnorm(gtmp/sig, log = TRUE)))
        }
        diffs = outer(as.vector(G), as.vector(f), FUN = "-")
        diffs.pos = diffs
        diffs.pos[diffs < 0] = NA
        diffs.neg = diffs
        diffs.neg[diffs >= 0] = NA
        idxu.mat = (diffs.pos == t(apply(diffs.pos, 2, min, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxl.mat = (diffs.neg == t(apply(diffs.neg, 2, max, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxu.mat[is.na(idxu.mat)] = FALSE
        idxl.mat[is.na(idxl.mat)] = FALSE
        G1 = (G %x% matrix(1, 1, N))[idxl.mat]
        G2 = (G %x% matrix(1, 1, N))[idxu.mat]
        g1 = (gtmp %x% matrix(1, 1, N))[idxl.mat]
        g2 = (gtmp %x% matrix(1, 1, N))[idxu.mat]
        beta1 = (G2 - G1)/(g2 - g1)
        g = (f - G1)/beta1 + g1
        g
    }
###
###  Setting up variables
###
    T = length(y)
    thetasave = matrix(NA, 5, ngibbs)
    s2ysave = matrix(NA, 1, ngibbs)
    mhy = 1
```

```
    mhtheta = matrix(1, 5, 1)
    thetatune = matrix(c(60, 0.3, 0.15, 0.01, 0.06), 5, 1)
    s2ytune = 10000
    nprec = 100
    TF1 = TRUE
###
###  Hyperpriors
###
    bounds = matrix(c(rep(0, 5), 400, 2, 1, 1, 1), 5, 2)
    mutheta = matrix(apply(bounds, 1, mean), ncol = 1)
    sigtheta = mutheta * 100
###
###  Starting values
###
    theta = matrix(c(220, 0.4, 0.8, 0.005, 0.46), ncol = 1)
    s2y = 10000
    j = 1
    x11(width = 10, height = 3)
    x11(width = 10, height = 3)
    x11(width = 10, height = 3)
    x11(width = 10, height = 3)
    x11(width = 10, height = 3)
    x11(width = 10, height = 3)
###
### Initialize timing variables
###
    time1 = proc.time()
    time2 = time1
    timeidx = 0
    hymod.time = 0
    opt.time = 0
###
###  Main Gibbs Loop
###
    for (k in 2:ngibbs) {
        cat(k, " ")
  ###
  ### Timing Calculations
  ###
        if (k == 12) {
            tentime = (proc.time() - time1)[3]
            cat("\n", tentime * (ngibbs/600), "expected minutes",
                "\n")
        }
        if (k%%100 == 0) {
            timeidx = timeidx + 1
            elapsetime = (proc.time() - time2)[3]
            cat("\n", elapsetime/60, "elapsed minutes", " ")
            leftidx = ngibbs/100 - timeidx
            cat(" ", (elapsetime/timeidx) * leftidx/60, "remaining minutes",
                "\n")
            cat((hymod.time/elapsetime) * 100, "percent of time in hymod",
                " ")
            cat(" ", (opt.time/elapsetime) * 100, "percent of time in optim",
```

```
            " ")
    }
###
###  Sample theta
###
    for (j in 1:5) {
        thetastar = theta
        thetastar[j, ] = rtnorm(1, theta[j, ], thetatune[j,
            ], bounds[j, 1], bounds[j, 2])
        hymod.time = hymod.time + system.time(fkminus1 <- hymod.C(x[,
            2], x[, 1], c(theta, 0, 0)))[3]
        hymod.time = hymod.time + system.time(fstar <- hymod.C(x[,
            2], x[, 1], c(thetastar, 0, 0)))[3]
        fkminus1 = matrix(fkminus1, ncol = 1)
        fstar = matrix(fstar, ncol = 1)
        opt.time = opt.time + system.time({
            gkminus1 <- getgvec(fkminus1, sqrt(s2y), nprec)
            gstar <- getgvec(fstar, sqrt(s2y), nprec)
        })[3]
        mhratio1 = sum(logdtnorm(y, gstar, sqrt(s2y))) +
            dtnorm(thetastar[j, ], mutheta[j, ], sigtheta[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE) +
            dtnorm(theta[j, ], thetastar[j, ], thetatune[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE)
        mhratio2 = sum(logdtnorm(y, gkminus1, sqrt(s2y))) +
            dtnorm(theta[j, ], mutheta[j, ], sigtheta[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE) +
            dtnorm(thetastar[j, ], theta[j, ], thetatune[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE)
        mhratio = exp(mhratio1 - mhratio2)
        if (mhratio > runif(1)) {
            theta = thetastar
            mhtheta[j, ] = mhtheta[j, ] + 1
            fkminus1 = fstar
        }
    }
###
###  Sample s2y
###
    s2ystar = rtnorm(1, s2y, s2ytune, 0)
    opt.time = opt.time + system.time(gstar <- getgvec(fkminus1,
        sqrt(s2ystar), nprec))[3]
    mhratio1 = sum(logdtnorm(y, gstar, sqrt(s2ystar))) +
        dnorm(log(s2y), s2mn, sqrt(s2var), log = T) + dtnorm(s2y,
        s2ystar, s2ytune, 0, log = TRUE)
    mhratio2 = sum(logdtnorm(y, gkminus1, sqrt(s2y))) + dnorm(log(s2ystar),
        s2mn, sqrt(s2var), log = T) + dtnorm(s2ystar, s2y,
        s2ytune, 0, log = TRUE)
    mhratio = exp(mhratio1 - mhratio2)
    if (mhratio > runif(1)) {
        s2y = s2ystar
        gkminus1 = gstar
        mhy = mhy + 1
    }
```

```
    ###
    ###  Saving Samples and Making Dynamic Trace Plots
    ###
          thetasave[, k] = theta
          s2ysave[, k] = s2y
          for (i in 1:5) {
              dev.set(i + 1)
              plot((thetasave[i, 1:k]), type = "l", main = i)
          }
          dev.set(7)
          plot((s2ysave[1, 1:k]), type = "l", main = "s2y")
    ###
    ###  Write out temporary output and get posterior predictive distribution
    ###
          if (k%%outdat == 0) {
              yhat = rtn(gkminus1, sqrt(s2y))
              if (TF1) {
                  fsave = cbind(fkminus1)
                  yhatsave = cbind(yhat)
              }
              if (!TF1) {
                  fsave = cbind(fsave, fkminus1)
                  yhatsave = cbind(yhatsave, yhat)
              }
              tmpout = list(ngibbs = ngibbs, mhtheta = mhtheta,
                  mhy = mhy, thetasave = thetasave, s2ysave = s2ysave,
                  yhatsave = yhatsave, fsave = fsave, y = y, x = x)
              save(tmpout, file = "tmpout.RData")
              TF1 = FALSE
          }
          cat("\n")
      }
###
###  Writing output
###
      cat("\n")
      list(ngibbs = ngibbs, mhtheta = mhtheta, mhy = mhy, thetasave = thetasave,
          s2ysave = s2ysave, opt.time = opt.time, hymod.time = hymod.time,
          yhatsave = yhatsave, y = y, x = x)
}
```

The previous code can be invoked using these commands:

```
data<-read.table("LeafRiverData.dat",header=F)
data<-data[3015:5205,-c(1:5)]
precip<-apply(data[,3:6],1,sum)
data<-cbind(data[,1:2],precip)
y<-matrix(data$V6,nrow=nrow(data))
X<-data[,3:2]
source("simpRRDAmcmc.R")
tmp.out=simpRRDAmcmc(y,X,9.3,100,10000)
```

SAC-SMA

Following is the code used to run the MCMC routine for SAC-SMA:

```
`sacramcmc` <-
function (y, PET, Precip, s2mn, s2var, ngibbs)
{
###
###  Subroutines
###
    require(msm)
    T = length(y)
    sedcommand = paste("sed 's/^.*#define MAXTSTEP.*$/#define MAXTSTEP ",
        T, "/'  model.h > temp.h", sep = "")
    system(sedcommand)
    system("mv temp.h model.h")
    system("rm sac_sma.so sac_sma.o")
    system("R CMD SHLIB sac_sma.c")
    dyn.load("sac_sma.so")
    sac_sma.C <- function(pet, precip, parameters) {
        .C("sac_sma", as.double(pet), as.double(c(t(precip))),
            as.double(parameters), runoff = as.double(rep(0,
                4 * length(pet))))$runoff
    }
    dinvgamma <- function(x, q, r) {
        1/(r^q * gamma(q)) * (x^(-(q + 1))) * exp(-1/(r * x))
    }
    logdinvgamma <- function(x, q, r) {
        -q * log(r) - lgamma(q) - (q + 1) * log(x) - 1/(r * x)
    }
    rtn <- function(mu, sig) {
        mu + sig * qnorm(log(runif(length(mu))) + pnorm(mu/sig,
            log = TRUE), lower.tail = FALSE, log.p = TRUE)
    }
    logdtnorm <- function(x, mean = 0, sd = 1) {
        lower = 0
        upper = Inf
        ret <- numeric(length(x))
        ret[x < lower | x > upper] <- 0
        ind <- x >= lower & x <= upper
        if (any(ind)) {
            xtmp <- dnorm(x, mean, sd, log = TRUE) - pnorm(mean/sd,
                log = TRUE)
            ret[x >= lower & x <= upper] <- xtmp[ind]
        }
        ret
    }
    getgvec <- function(f, sig, n) {
        l = -100
        N = length(f)
        u = seq(1e-11, 0.999999999999, , n)
```

```
        gtmp = sig * qnorm(u, , sig)
        n = n + 1
        gtmp = c(gtmp, max(f))
        G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) - pnorm(gtmp/sig,
            log = TRUE)))
        l = min(gtmp)
        while (min(G) >= min(f)) {
            l = l - 100
            gtmp = c(l, gtmp)
            n = n + 1
            G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) -
                pnorm(gtmp/sig, log = TRUE)))
        }
        diffs = outer(as.vector(G), as.vector(f), FUN = "-")
        diffs.pos = diffs
        diffs.pos[diffs < 0] = NA
        diffs.neg = diffs
        diffs.neg[diffs >= 0] = NA
        idxu.mat = (diffs.pos == t(apply(diffs.pos, 2, min, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxl.mat = (diffs.neg == t(apply(diffs.neg, 2, max, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxu.mat[is.na(idxu.mat)] = FALSE
        idxl.mat[is.na(idxl.mat)] = FALSE
        G1 = (G %x% matrix(1, 1, N))[idxl.mat]
        G2 = (G %x% matrix(1, 1, N))[idxu.mat]
        g1 = (gtmp %x% matrix(1, 1, N))[idxl.mat]
        g2 = (gtmp %x% matrix(1, 1, N))[idxu.mat]
        beta1 = (G2 - G1)/(g2 - g1)
        g = (f - G1)/beta1 + g1
        g
    }
###
###  Setting up variables
###
    outdat = 10
    uh <- c(1.4, 3.2, 4.5, 5.1, 5.2, 5.4, 6.1, 6.9, 7.3, 7.3,
        7.1, 6.8, 6.1, 5.2, 4.2, 3.4, 2.6, 1.6, 0.6)
    thetasave = matrix(NA, 23, ngibbs)
    s2ysave = matrix(NA, 1, ngibbs)
    mhy = 1
    mhtheta = matrix(1, 23, 1)
    thetatune = matrix(c(15, 20, 0.05, 0.01, 1000, 1000, 25,
        0.5, 60, 150, 100, 0.05, 0.005, 0.075, 1000, 1000, 1000,
        1000, 1000, 1000, 1000, 1000, 1000), 23, 1)
    s2ytune = 1000
    nprec = 100
    TF1 = TRUE
###
###  Hyperpriors
###
    bounds = matrix(c(1, 150, 1, 150, 0.1, 0.5, 0, 0.1, 0, 0.4,
        0, 1, 1, 250, 1, 5, 1, 500, 1, 1000, 1, 1000, 0.01, 0.25,
        1e-04, 0.025, 0, 0.6, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
```

```
        1, 0, 1, 0, 1, 0, 1), 23, 2, byrow = T)
    mutheta = matrix(apply(bounds, 1, mean), ncol = 1)
    sigtheta = mutheta * 100
###
###  Starting values
###
    theta = matrix(c(58, 45.8, 0.18, 1e-05, 0.4, 0, 230.2, 3.2,
        247.5, 40.5, 124.5, 0.18, 0.006, 0.04, 0.3, 0, 1, 0.5,
        0.5, 0.5, 0.5, 0.5, 0.5), ncol = 1)
    s2y = 500
    j = 1
    for (i in 1:13) {
        x11(width = 10, height = 3)
    }
###
### Initialize timing variables
###
    time1 = proc.time()
    time2 = time1
    timeidx = 0
###
###  Main Gibbs Loop
###
    for (k in 2:ngibbs) {
        cat(k, " ")
  ###
  ### Timing Calculations
  ###
        if (k == 12) {
            tentime = (proc.time() - time1)[3]
            cat("\n", tentime * (ngibbs/600), "expected minutes",
                "\n")
        }
        if (k%%100 == 0) {
            timeidx = timeidx + 1
            elapsetime = (proc.time() - time2)[3]
            cat("\n", elapsetime/60, "elapsed minutes", " ")
            leftidx = ngibbs/100 - timeidx
            cat(" ", (elapsetime/timeidx) * leftidx/60, "remaining minutes",
                "\n")
        }
  ###
  ###  Sample theta
  ###
        for (j in c(1:23)[-c(5, 6, 15:23)]) {
            thetastar = theta
            thetastar[j, ] = rtnorm(1, theta[j, ], thetatune[j,
                ], bounds[j, 1], bounds[j, 2])
            fkminus1 <- sac_sma.C(PET, Precip, theta)
            fkminus1 <- convolve(fkminus1, rev(uh), type = "o")[-seq(from = length(fkminus1),
                by = -1, length = 18)]
            fkminus1 <- matrix(fkminus1, ncol = 4, byrow = T) %*%
                rep(0.25, 4)
            fstar <- sac_sma.C(PET, Precip, thetastar)
```

```
        fstar <- convolve(fstar, rev(uh), type = "o")[-seq(from = length(fstar),
            by = -1, length = 18)]
        fstar <- matrix(fstar, ncol = 4, byrow = T) %*% rep(0.25,
            4)
        if (any(c(fkminus1, fstar) == 0)) {
            cat("zero")
        }
        gkminus1 <- getgvec(fkminus1, sqrt(s2y), nprec)
        gstar <- getgvec(fstar, sqrt(s2y), nprec)
        mhratio1 = sum(logdtnorm(y, gstar, sqrt(s2y))) +
            dtnorm(thetastar[j, ], mutheta[j, ], sigtheta[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE) +
            dtnorm(theta[j, ], thetastar[j, ], thetatune[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE)
        mhratio2 = sum(logdtnorm(y, gkminus1, sqrt(s2y))) +
            dtnorm(theta[j, ], mutheta[j, ], sigtheta[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE) +
            dtnorm(thetastar[j, ], theta[j, ], thetatune[j,
              ], bounds[j, 1], bounds[j, 2], log = TRUE)
        mhratio = exp(mhratio1 - mhratio2)
        if (mhratio > runif(1)) {
            theta = thetastar
            mhtheta[j, ] = mhtheta[j, ] + 1
            fkminus1 = fstar
        }
    }
###
###  Sample s2y
###
    s2ystar = rtnorm(1, s2y, s2ytune, 0)
    gstar <- getgvec(fkminus1, sqrt(s2ystar), nprec)
    mhratio1 = sum(logdtnorm(y, gstar, sqrt(s2ystar))) +
        dnorm(log(s2y), s2mn, sqrt(s2var), log = T) + dtnorm(s2y,
        s2ystar, s2ytune, 0, log = TRUE)
    mhratio2 = sum(logdtnorm(y, gkminus1, sqrt(s2y))) + dnorm(log(s2ystar),
        s2mn, sqrt(s2var), log = T) + dtnorm(s2ystar, s2y,
        s2ytune, 0, log = TRUE)
    mhratio = exp(mhratio1 - mhratio2)
    if (mhratio > runif(1)) {
        s2y = s2ystar
        gkminus1 = gstar
        mhy = mhy + 1
    }
###
###  Saving Samples and Making Dynamic Trace Plots
###
    thetasave[, k] = theta
    s2ysave[, k] = s2y
    index <- c(1:23)[-c(5, 6, 15:23)]
    for (i in 1:12) {
        dev.set(i + 1)
        plot((thetasave[index[i], 1:k]), type = "l", main = index[i])
    }
    dev.set(14)
```

```
        plot((s2ysave[1, 1:k]), type = "l", main = "s2y")
    ###
    ###  Write out temporary output and get posterior predictive distribution
    ###
        if (k%%outdat == 0) {
            yhat = rtn(gkminus1, sqrt(s2y))
            if (TF1) {
                fsave = cbind(fkminus1)
                yhatsave = cbind(yhat)
            }
            if (!TF1) {
                fsave = cbind(fsave, fkminus1)
                yhatsave = cbind(yhatsave, yhat)
            }
            tmpout = list(ngibbs = ngibbs, mhtheta = mhtheta,
                mhy = mhy, thetasave = thetasave, s2ysave = s2ysave,
                yhatsave = yhatsave, fsave = fsave, y = y, PET = PET,
                Precip = Precip)
            save(tmpout, file = "tmpout.RData")
            TF1 = FALSE
        }
        cat("\n")
    }
    cat("\n")
###
###  Writing output
###
    list(ngibbs = ngibbs, mhtheta = mhtheta, mhy = mhy, thetasave = thetasave,
        s2ysave = s2ysave, yhatsave = yhatsave, fsave = fsave,
        y = y, PET = PET, Precip = Precip)
}
```

The previous code can be invoked using these commands:

```
data<-read.table("LeafRiverData.dat",header=F)
data<-data[3015:5205,-c(1:5)]
rfro<-matrix(data[,1],nrow=nrow(data))
evap<-matrix(data[,2],nrow=nrow(data))
rain<-as.matrix(data[,3:6])
source("sacramcmc.R")
tmp.out=sacramcmc(rfro,evap,rain,9.3,100,10000)
```

"ψ_t, z_t" Model

Following is the code used to run the MCMC routine for the "ψ_t, z_t" Model.

```
`rrbma` <-
function (h.out, s.out)
{
###
###  Subroutines
###
    require(msm)
    getgvec <- function(f, sig, n) {
        l = -100
        N = length(f)
        u = seq(1e-11, 0.999999999999, , n)
        gtmp = sig * qnorm(u, , sig)
        n = n + 1
        gtmp = c(gtmp, max(f))
        G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) - pnorm(gtmp/sig,
            log = TRUE)))
        l = min(gtmp)
        while (min(G) >= min(f)) {
            l = l - 100
            gtmp = c(l, gtmp)
            n = n + 1
            G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) -
                pnorm(gtmp/sig, log = TRUE)))
        }
        diffs = outer(as.vector(G), as.vector(f), FUN = "-")
        diffs.pos = diffs
        diffs.pos[diffs < 0] = NA
        diffs.neg = diffs
        diffs.neg[diffs >= 0] = NA
        idxu.mat = (diffs.pos == t(apply(diffs.pos, 2, min, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxl.mat = (diffs.neg == t(apply(diffs.neg, 2, max, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxu.mat[is.na(idxu.mat)] = FALSE
        idxl.mat[is.na(idxl.mat)] = FALSE
        G1 = (G %x% matrix(1, 1, N))[idxl.mat]
        G2 = (G %x% matrix(1, 1, N))[idxu.mat]
        g1 = (gtmp %x% matrix(1, 1, N))[idxl.mat]
        g2 = (gtmp %x% matrix(1, 1, N))[idxu.mat]
        beta1 = (G2 - G1)/(g2 - g1)
        g = (f - G1)/beta1 + g1
        G = g + sig * (exp(dnorm(g/sig, log = TRUE) - pnorm(g/sig,
            log = TRUE)))
        G.neg = G
        idxNEG = (G <= 0)
        G[idxNEG] = min(G[!idxNEG])
        g[idxNEG] = min(g[!idxNEG])
        g
    }
    rtn <- function(mu, sig) {
        mu + sig * qnorm(log(runif(length(mu))) + pnorm(mu/sig,
            log = TRUE), lower.tail = FALSE, log.p = TRUE)
    }
    logdtnorm <- function(x, mean = 0, sd = 1) {
```

```
        lower = 0
        upper = Inf
        ret <- numeric(length(x))
        ret[x < lower | x > upper] <- 0
        ind <- x >= lower & x <= upper
        if (any(ind)) {
            xtmp <- dnorm(x, mean, sd, log = TRUE) - pnorm(mean/sd,
                log = TRUE)
            ret[x >= lower & x <= upper] <- xtmp[ind]
        }
        ret
    }
###
###  Initialize Variables
###
    ngibbs = h.out$ngibbs
    nkeep = dim(h.out$yhatsave)[2]
    y = h.out$y
    cat(ngibbs, nkeep, length(y), "\n")
    f.h = h.out$fsave
    f.s = s.out$fsave
    g.h.mn = rep(0, length(y))
    g.s.mn = rep(0, length(y))
    g.mn = rep(0, length(y))
    Dhat.avg.h = 0
    Dhat.avg.s = 0
    Dhat.avg = 0
    s2y.mn = 0
    s2y.s = s.out$s2ysave[, seq(ngibbs/nkeep, ngibbs, ngibbs/nkeep)]
    s2y.h = h.out$s2ysave[, seq(ngibbs/nkeep, ngibbs, ngibbs/nkeep)]
    yhatsave = matrix(0, length(y), nkeep)
    psave = matrix(0, 1, nkeep)
    zsave = matrix(0, 1, nkeep)
###
###  Priors and Starting Values
###
    a = 1
    b = 1
    z = 1
    p = 0.5
###
###  Begin Gibbs Loop
###
    for (k in 1:nkeep) {
        cat(k, " ")
  ###
  ###  Get g for hymod
  ###
        g.h = getgvec(f.h[, k], sqrt(s2y.h[k]), 100)
  ###
  ###  Get g for sac
  ###
        g.s = getgvec(f.s[, k], sqrt(s2y.s[k]), 100)
  ###
```

```
  ###  Sample p
  ###
       p = rbeta(1, z + a, (1 - z) + b)
  ###
  ###  Sample z
  ###
       tmpp = 1/(1 + exp(log(1 - p) + sum(logdtnorm(y, g.s,
           sqrt(s2y.s[k]))) - log(p) - sum(logdtnorm(y, g.h,
           sqrt(s2y.h[k])))))
       z = rbinom(1, 1, tmpp)
  ###
  ###  Get Predictions
  ###
       g = z * g.h + (1 - z) * g.s
       s2y = z * s2y.h[k] + (1 - z) * s2y.s[k]
       s2y.mn = s2y.mn + s2y/nkeep
       yhatsave[, k] = rtn(g, sqrt(s2y))
  ###
  ###  DIC calculations
  ###
       g.h.mn = g.h.mn + g.h/nkeep
       g.s.mn = g.s.mn + g.s/nkeep
       g.mn = g.mn + g/nkeep
       Dhat.avg.h = Dhat.avg.h + (-2 * sum(logdtnorm(y, g.h,
           sqrt(s2y.h[k]))))/nkeep
       Dhat.avg.s = Dhat.avg.s + (-2 * sum(logdtnorm(y, g.s,
           sqrt(s2y.s[k]))))/nkeep
       Dhat.avg = Dhat.avg + (-2 * sum(logdtnorm(y, g, sqrt(s2y))))/nkeep
  ###
  ###  Save Samples
  ###
       psave[, k] = p
       zsave[, k] = z
    }
    cat("\n")
###
###  DIC Calculations
###
    D.thetahat.h = -2 * sum(logdtnorm(y, g.h.mn, sqrt(mean(s2y.h))))
    D.thetahat.s = -2 * sum(logdtnorm(y, g.s.mn, sqrt(mean(s2y.s))))
    D.thetahat = -2 * sum(logdtnorm(y, g.mn, sqrt(mean(s2y.mn))))
    DIC.h = 2 * Dhat.avg.h - D.thetahat.h
    DIC.s = 2 * Dhat.avg.s - D.thetahat.s
    DIC = 2 * Dhat.avg - D.thetahat
    pD.h = Dhat.avg.h - D.thetahat.h
    pD.s = Dhat.avg.s - D.thetahat.s
    pD = Dhat.avg - D.thetahat
    rmse.h = sqrt(t(apply(h.out$yhatsave, 1, mean) - y) %*% (apply(h.out$yhatsave,
        1, mean) - y))
    rmse.s = sqrt(t(apply(s.out$yhatsave, 1, mean) - y) %*% (apply(s.out$yhatsave,
        1, mean) - y))
    rmse = sqrt(t(apply(yhatsave, 1, mean) - y) %*% (apply(yhatsave,
        1, mean) - y))
###
```

```
###  Print DIC Output
###
    DIC.out = matrix(c(pD.h, pD.s, pD, DIC.h, DIC.s, DIC, Dhat.avg.h,
        Dhat.avg.s, Dhat.avg, D.thetahat.h, D.thetahat.s, D.thetahat,
        rmse.h, rmse.s, rmse), 3, 5)
    DIC.df = data.frame(DIC.out)
    row.names(DIC.df) = c("HYMOD", "SAC", "Mixture")
    names(DIC.df) = c("pD", "DIC", "Dhat.avg", "D.thetahat",
        "rmse")
    print(DIC.df)
###
###  Write Output
###
    list(ngibbs = ngibbs, yhatsave = yhatsave, psave = psave,
        zsave = zsave, y = y, pD = pD, pD.h = pD.h, pD.s = pD.s,
        DIC = DIC, DIC.h = DIC.h, DIC.s = DIC.s, D.thetahat.h = D.thetahat.h,
        D.thetahat.s = D.thetahat.s, D.thetahat = D.thetahat,
        Dhat.avg.h = Dhat.avg.h, Dhat.avg.s = Dhat.avg.s, Dhat.avg = Dhat.avg,
        g.h.mn = g.h.mn, g.s.mn = g.s.mn, g.mn = g.mn, rmse = rmse,
        rmse.h = rmse.h, rmse.s = rmse.s, DIC.out = DIC.out,
        DIC.df = DIC.df)
}
```

"ψ, z_t" Model

Following is the code used to run the MCMC routine for the "ψ_t, z_t" Model.

```
`rrbma` <-
function (h.out, s.out)
{
###
###  Subroutines
###
    require(msm)
    getgvec <- function(f, sig, n) {
        l = -100
        N = length(f)
        u = seq(1e-11, 0.999999999999, , n)
        gtmp = sig * qnorm(u, , sig)
        n = n + 1
        gtmp = c(gtmp, max(f))
        G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) - pnorm(gtmp/sig,
            log = TRUE)))
        l = min(gtmp)
        while (min(G) >= min(f)) {
            l = l - 100
            gtmp = c(l, gtmp)
            n = n + 1
            G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) -
```

```
            pnorm(gtmp/sig, log = TRUE)))
        }
        diffs = outer(as.vector(G), as.vector(f), FUN = "-")
        diffs.pos = diffs
        diffs.pos[diffs < 0] = NA
        diffs.neg = diffs
        diffs.neg[diffs >= 0] = NA
        idxu.mat = (diffs.pos == t(apply(diffs.pos, 2, min, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxl.mat = (diffs.neg == t(apply(diffs.neg, 2, max, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxu.mat[is.na(idxu.mat)] = FALSE
        idxl.mat[is.na(idxl.mat)] = FALSE
        G1 = (G %x% matrix(1, 1, N))[idxl.mat]
        G2 = (G %x% matrix(1, 1, N))[idxu.mat]
        g1 = (gtmp %x% matrix(1, 1, N))[idxl.mat]
        g2 = (gtmp %x% matrix(1, 1, N))[idxu.mat]
        beta1 = (G2 - G1)/(g2 - g1)
        g = (f - G1)/beta1 + g1
        G = g + sig * (exp(dnorm(g/sig, log = TRUE) - pnorm(g/sig,
            log = TRUE)))
        G.neg = G
        idxNEG = (G <= 0)
        G[idxNEG] = min(G[!idxNEG])
        g[idxNEG] = min(g[!idxNEG])
        g
    }
    rtn <- function(mu, sig) {
        mu + sig * qnorm(log(runif(length(mu))) + pnorm(mu/sig,
            log = TRUE), lower.tail = FALSE, log.p = TRUE)
    }
    logdtnorm <- function(x, mean = 0, sd = 1) {
        lower = 0
        upper = Inf
        ret <- numeric(length(x))
        ret[x < lower | x > upper] <- 0
        ind <- x >= lower & x <= upper
        if (any(ind)) {
            xtmp <- dnorm(x, mean, sd, log = TRUE) - pnorm(mean/sd,
                log = TRUE)
            ret[x >= lower & x <= upper] <- xtmp[ind]
        }
        ret
    }
###
###  Initialize Variables
###
    ngibbs = h.out$ngibbs
    nkeep = dim(h.out$yhatsave)[2]
    y = h.out$y
    cat(ngibbs, nkeep, length(y), "\n")
    f.h = h.out$fsave
    f.s = s.out$fsave
    g.h.mn = rep(0, length(y))
```

```
    g.s.mn = rep(0, length(y))
    g.mn = rep(0, length(y))
    Dhat.avg.h = 0
    Dhat.avg.s = 0
    Dhat.avg = 0
    s2y.mn = 0
    s2y.s = s.out$s2ysave[, seq(ngibbs/nkeep, ngibbs, ngibbs/nkeep)]
    s2y.h = h.out$s2ysave[, seq(ngibbs/nkeep, ngibbs, ngibbs/nkeep)]
    yhatsave = matrix(0, length(y), nkeep)
    psave = matrix(0, 1, nkeep)
    zsave = matrix(0, 1, nkeep)
###
###  Priors and Starting Values
###
    a = 1
    b = 1
    z = 1
    p = 0.5
###
###  Begin Gibbs Loop
###
    for (k in 1:nkeep) {
        cat(k, " ")
  ###
  ###  Get g for hymod
  ###
        g.h = getgvec(f.h[, k], sqrt(s2y.h[k]), 100)
  ###
  ###  Get g for sac
  ###
        g.s = getgvec(f.s[, k], sqrt(s2y.s[k]), 100)
  ###
  ###  Sample p
  ###
        p = rbeta(1, z + a, (1 - z) + b)
  ###
  ###  Sample z
  ###
        tmpp = 1/(1 + exp(log(1 - p) + sum(logdtnorm(y, g.s,
            sqrt(s2y.s[k]))) - log(p) - sum(logdtnorm(y, g.h,
            sqrt(s2y.h[k])))))
        z = rbinom(1, 1, tmpp)
  ###
  ###  Get Predictions
  ###
        g = z * g.h + (1 - z) * g.s
        s2y = z * s2y.h[k] + (1 - z) * s2y.s[k]
        s2y.mn = s2y.mn + s2y/nkeep
        yhatsave[, k] = rtn(g, sqrt(s2y))
  ###
  ###  DIC calculations
  ###
        g.h.mn = g.h.mn + g.h/nkeep
        g.s.mn = g.s.mn + g.s/nkeep
```

```
        g.mn = g.mn + g/nkeep
        Dhat.avg.h = Dhat.avg.h + (-2 * sum(logdtnorm(y, g.h,
            sqrt(s2y.h[k]))))/nkeep
        Dhat.avg.s = Dhat.avg.s + (-2 * sum(logdtnorm(y, g.s,
            sqrt(s2y.s[k]))))/nkeep
        Dhat.avg = Dhat.avg + (-2 * sum(logdtnorm(y, g, sqrt(s2y))))/nkeep
  ###
  ###  Save Samples
  ###
        psave[, k] = p
        zsave[, k] = z
    }
    cat("\n")
###
###  DIC Calculations
###
    D.thetahat.h = -2 * sum(logdtnorm(y, g.h.mn, sqrt(mean(s2y.h))))
    D.thetahat.s = -2 * sum(logdtnorm(y, g.s.mn, sqrt(mean(s2y.s))))
    D.thetahat = -2 * sum(logdtnorm(y, g.mn, sqrt(mean(s2y.mn))))
    DIC.h = 2 * Dhat.avg.h - D.thetahat.h
    DIC.s = 2 * Dhat.avg.s - D.thetahat.s
    DIC = 2 * Dhat.avg - D.thetahat
    pD.h = Dhat.avg.h - D.thetahat.h
    pD.s = Dhat.avg.s - D.thetahat.s
    pD = Dhat.avg - D.thetahat
    rmse.h = sqrt(t(apply(h.out$yhatsave, 1, mean) - y) %*% (apply(h.out$yhatsave,
        1, mean) - y))
    rmse.s = sqrt(t(apply(s.out$yhatsave, 1, mean) - y) %*% (apply(s.out$yhatsave,
        1, mean) - y))
    rmse = sqrt(t(apply(yhatsave, 1, mean) - y) %*% (apply(yhatsave,
        1, mean) - y))
###
###  Print DIC Output
###
    DIC.out = matrix(c(pD.h, pD.s, pD, DIC.h, DIC.s, DIC, Dhat.avg.h,
        Dhat.avg.s, Dhat.avg, D.thetahat.h, D.thetahat.s, D.thetahat,
        rmse.h, rmse.s, rmse), 3, 5)
    DIC.df = data.frame(DIC.out)
    row.names(DIC.df) = c("HYMOD", "SAC", "Mixture")
    names(DIC.df) = c("pD", "DIC", "Dhat.avg", "D.thetahat",
        "rmse")
    print(DIC.df)
###
###  Write Output
###
    list(ngibbs = ngibbs, yhatsave = yhatsave, psave = psave,
        zsave = zsave, y = y, pD = pD, pD.h = pD.h, pD.s = pD.s,
        DIC = DIC, DIC.h = DIC.h, DIC.s = DIC.s, D.thetahat.h = D.thetahat.h,
        D.thetahat.s = D.thetahat.s, D.thetahat = D.thetahat,
        Dhat.avg.h = Dhat.avg.h, Dhat.avg.s = Dhat.avg.s, Dhat.avg = Dhat.avg,
        g.h.mn = g.h.mn, g.s.mn = g.s.mn, g.mn = g.mn, rmse = rmse,
        rmse.h = rmse.h, rmse.s = rmse.s, DIC.out = DIC.out,
        DIC.df = DIC.df)
}
```

"ψ, z" Model

Following is the code used to run the MCMC routine for the "ψ_t, z_t" Model.

```
`rrbma` <-
function (h.out, s.out)
{
###
###  Subroutines
###
    require(msm)
    getgvec <- function(f, sig, n) {
        l = -100
        N = length(f)
        u = seq(1e-11, 0.999999999999, , n)
        gtmp = sig * qnorm(u, , sig)
        n = n + 1
        gtmp = c(gtmp, max(f))
        G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) - pnorm(gtmp/sig,
            log = TRUE)))
        l = min(gtmp)
        while (min(G) >= min(f)) {
            l = l - 100
            gtmp = c(l, gtmp)
            n = n + 1
            G = gtmp + sig * (exp(dnorm(gtmp/sig, log = TRUE) -
                pnorm(gtmp/sig, log = TRUE)))
        }
        diffs = outer(as.vector(G), as.vector(f), FUN = "-")
        diffs.pos = diffs
        diffs.pos[diffs < 0] = NA
        diffs.neg = diffs
        diffs.neg[diffs >= 0] = NA
        idxu.mat = (diffs.pos == t(apply(diffs.pos, 2, min, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxl.mat = (diffs.neg == t(apply(diffs.neg, 2, max, na.rm = TRUE) %x%
            matrix(1, 1, n)))
        idxu.mat[is.na(idxu.mat)] = FALSE
        idxl.mat[is.na(idxl.mat)] = FALSE
        G1 = (G %x% matrix(1, 1, N))[idxl.mat]
        G2 = (G %x% matrix(1, 1, N))[idxu.mat]
        g1 = (gtmp %x% matrix(1, 1, N))[idxl.mat]
        g2 = (gtmp %x% matrix(1, 1, N))[idxu.mat]
        beta1 = (G2 - G1)/(g2 - g1)
        g = (f - G1)/beta1 + g1
        G = g + sig * (exp(dnorm(g/sig, log = TRUE) - pnorm(g/sig,
            log = TRUE)))
        G.neg = G
        idxNEG = (G <= 0)
        G[idxNEG] = min(G[!idxNEG])
        g[idxNEG] = min(g[!idxNEG])
```

```
        g
    }
    rtn <- function(mu, sig) {
        mu + sig * qnorm(log(runif(length(mu))) + pnorm(mu/sig,
            log = TRUE), lower.tail = FALSE, log.p = TRUE)
    }
    logdtnorm <- function(x, mean = 0, sd = 1) {
        lower = 0
        upper = Inf
        ret <- numeric(length(x))
        ret[x < lower | x > upper] <- 0
        ind <- x >= lower & x <= upper
        if (any(ind)) {
            xtmp <- dnorm(x, mean, sd, log = TRUE) - pnorm(mean/sd,
                log = TRUE)
            ret[x >= lower & x <= upper] <- xtmp[ind]
        }
        ret
    }
###
###  Initialize Variables
###
    ngibbs = h.out$ngibbs
    nkeep = dim(h.out$yhatsave)[2]
    y = h.out$y
    cat(ngibbs, nkeep, length(y), "\n")
    f.h = h.out$fsave
    f.s = s.out$fsave
    g.h.mn = rep(0, length(y))
    g.s.mn = rep(0, length(y))
    g.mn = rep(0, length(y))
    Dhat.avg.h = 0
    Dhat.avg.s = 0
    Dhat.avg = 0
    s2y.mn = 0
    s2y.s = s.out$s2ysave[, seq(ngibbs/nkeep, ngibbs, ngibbs/nkeep)]
    s2y.h = h.out$s2ysave[, seq(ngibbs/nkeep, ngibbs, ngibbs/nkeep)]
    yhatsave = matrix(0, length(y), nkeep)
    psave = matrix(0, 1, nkeep)
    zsave = matrix(0, 1, nkeep)
###
###  Priors and Starting Values
###
    a = 1
    b = 1
    z = 1
    p = 0.5
###
###  Begin Gibbs Loop
###
    for (k in 1:nkeep) {
        cat(k, " ")
  ###
  ###  Get g for hymod
```

```
  ###
        g.h = getgvec(f.h[, k], sqrt(s2y.h[k]), 100)
  ###
  ###  Get g for sac
  ###
        g.s = getgvec(f.s[, k], sqrt(s2y.s[k]), 100)
  ###
  ###  Sample p
  ###
        p = rbeta(1, z + a, (1 - z) + b)
  ###
  ###  Sample z
  ###
        tmpp = 1/(1 + exp(log(1 - p) + sum(logdtnorm(y, g.s,
            sqrt(s2y.s[k]))) - log(p) - sum(logdtnorm(y, g.h,
            sqrt(s2y.h[k])))))
        z = rbinom(1, 1, tmpp)
  ###
  ###  Get Predictions
  ###
        g = z * g.h + (1 - z) * g.s
        s2y = z * s2y.h[k] + (1 - z) * s2y.s[k]
        s2y.mn = s2y.mn + s2y/nkeep
        yhatsave[, k] = rtn(g, sqrt(s2y))
  ###
  ###  DIC calculations
  ###
        g.h.mn = g.h.mn + g.h/nkeep
        g.s.mn = g.s.mn + g.s/nkeep
        g.mn = g.mn + g/nkeep
        Dhat.avg.h = Dhat.avg.h + (-2 * sum(logdtnorm(y, g.h,
            sqrt(s2y.h[k]))))/nkeep
        Dhat.avg.s = Dhat.avg.s + (-2 * sum(logdtnorm(y, g.s,
            sqrt(s2y.s[k]))))/nkeep
        Dhat.avg = Dhat.avg + (-2 * sum(logdtnorm(y, g, sqrt(s2y))))/nkeep
  ###
  ###  Save Samples
  ###
        psave[, k] = p
        zsave[, k] = z
    }
    cat("\n")
###
###  DIC Calculations
###
    D.thetahat.h = -2 * sum(logdtnorm(y, g.h.mn, sqrt(mean(s2y.h))))
    D.thetahat.s = -2 * sum(logdtnorm(y, g.s.mn, sqrt(mean(s2y.s))))
    D.thetahat = -2 * sum(logdtnorm(y, g.mn, sqrt(mean(s2y.mn))))
    DIC.h = 2 * Dhat.avg.h - D.thetahat.h
    DIC.s = 2 * Dhat.avg.s - D.thetahat.s
    DIC = 2 * Dhat.avg - D.thetahat
    pD.h = Dhat.avg.h - D.thetahat.h
    pD.s = Dhat.avg.s - D.thetahat.s
    pD = Dhat.avg - D.thetahat
```

```
    rmse.h = sqrt(t(apply(h.out$yhatsave, 1, mean) - y) %*% (apply(h.out$yhatsave,
        1, mean) - y))
    rmse.s = sqrt(t(apply(s.out$yhatsave, 1, mean) - y) %*% (apply(s.out$yhatsave,
        1, mean) - y))
    rmse = sqrt(t(apply(yhatsave, 1, mean) - y) %*% (apply(yhatsave,
        1, mean) - y))
###
###  Print DIC Output
###
    DIC.out = matrix(c(pD.h, pD.s, pD, DIC.h, DIC.s, DIC, Dhat.avg.h,
        Dhat.avg.s, Dhat.avg, D.thetahat.h, D.thetahat.s, D.thetahat,
        rmse.h, rmse.s, rmse), 3, 5)
    DIC.df = data.frame(DIC.out)
    row.names(DIC.df) = c("HYMOD", "SAC", "Mixture")
    names(DIC.df) = c("pD", "DIC", "Dhat.avg", "D.thetahat",
        "rmse")
    print(DIC.df)
###
###  Write Output
###
    list(ngibbs = ngibbs, yhatsave = yhatsave, psave = psave,
        zsave = zsave, y = y, pD = pD, pD.h = pD.h, pD.s = pD.s,
        DIC = DIC, DIC.h = DIC.h, DIC.s = DIC.s, D.thetahat.h = D.thetahat.h,
        D.thetahat.s = D.thetahat.s, D.thetahat = D.thetahat,
        Dhat.avg.h = Dhat.avg.h, Dhat.avg.s = Dhat.avg.s, Dhat.avg = Dhat.avg,
        g.h.mn = g.h.mn, g.s.mn = g.s.mn, g.mn = g.mn, rmse = rmse,
        rmse.h = rmse.h, rmse.s = rmse.s, DIC.out = DIC.out,
        DIC.df = DIC.df)
}
```

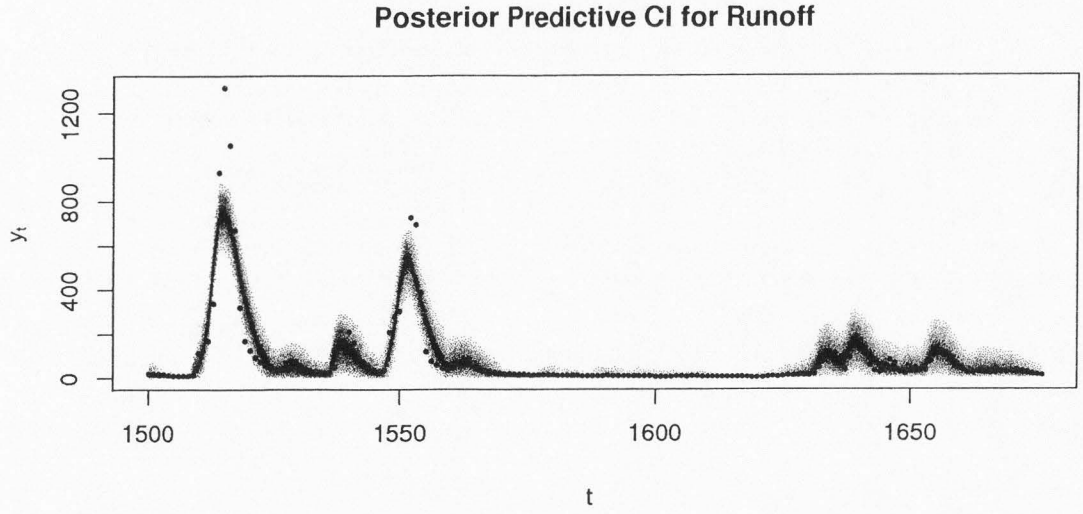

Figure 1: Posterior predictive distribuion of rainfall runoff using HYMOD. The shaded black area is the posterior predictive distribution of runoff and the points are the measured values of runoff. Units for runoff are cubic meters per second days (cmsd) and units for time are days.

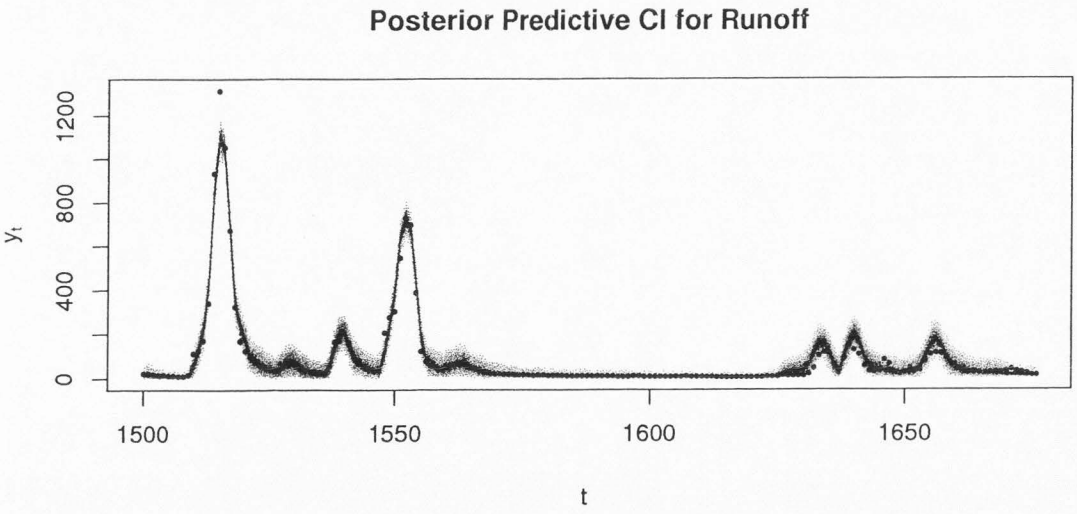

Figure 2: Posterior predictive distribuion of rainfall runoff using SAC-SMA. The shaded black area is the posterior predictive distribution of runoff and the points are the measured values of runoff. Units for runoff are cubic meters per second days (cmsd) and units for time are days.

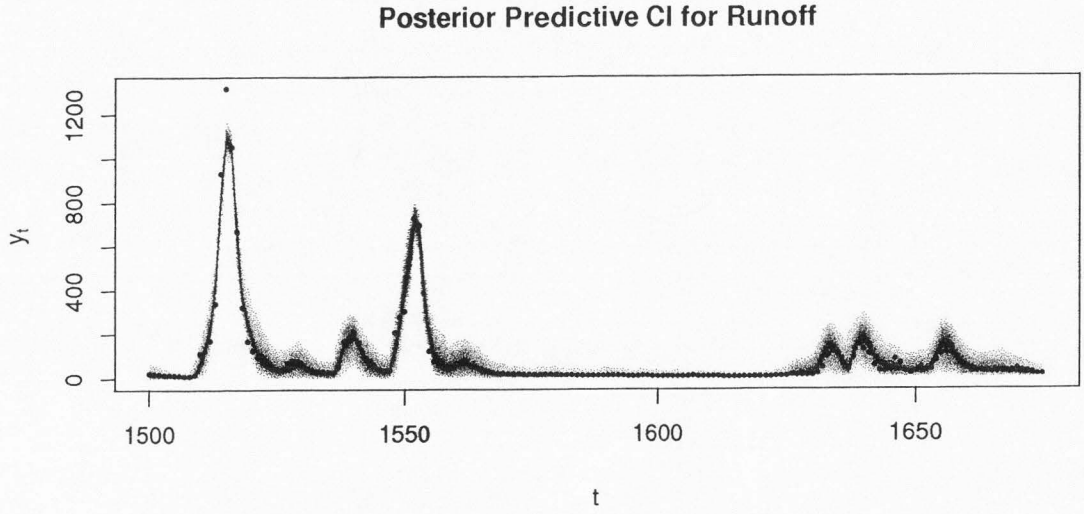

Figure 3: Posterior predictive distribuion of rainfall runoff using the "ψ_t, z_t" model. The shaded black area is the posterior predictive distribution of runoff and the points are the measured values of runoff. Units for runoff are cubic meters per second days (cmsd) and units for time are days.

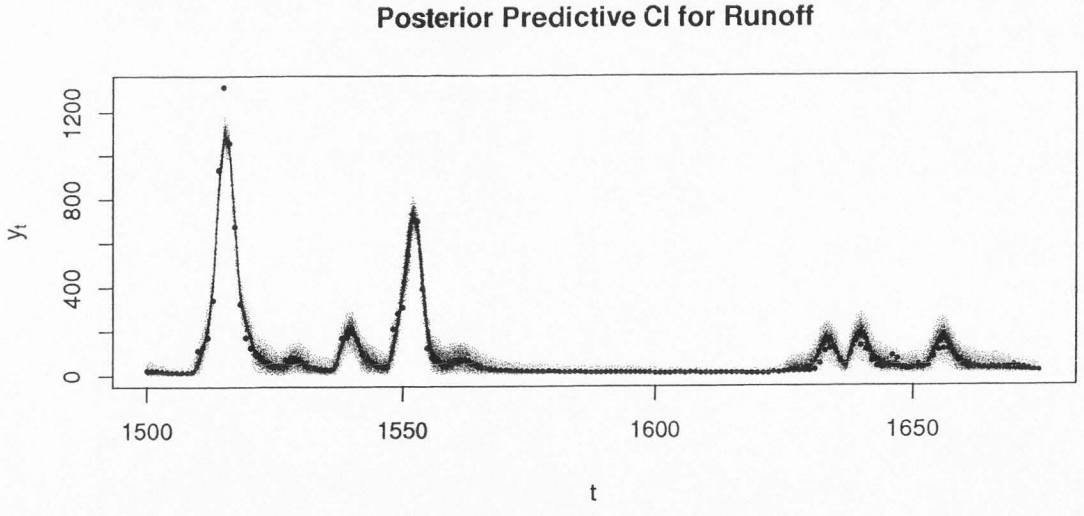

Figure 4: Posterior predictive distribuion of rainfall runoff using the "ψ, z_t" model. The shaded black area is the posterior predictive distribution of runoff and the points are the measured values of runoff. Units for runoff are cubic meters per second days (cmsd) and units for time are days.

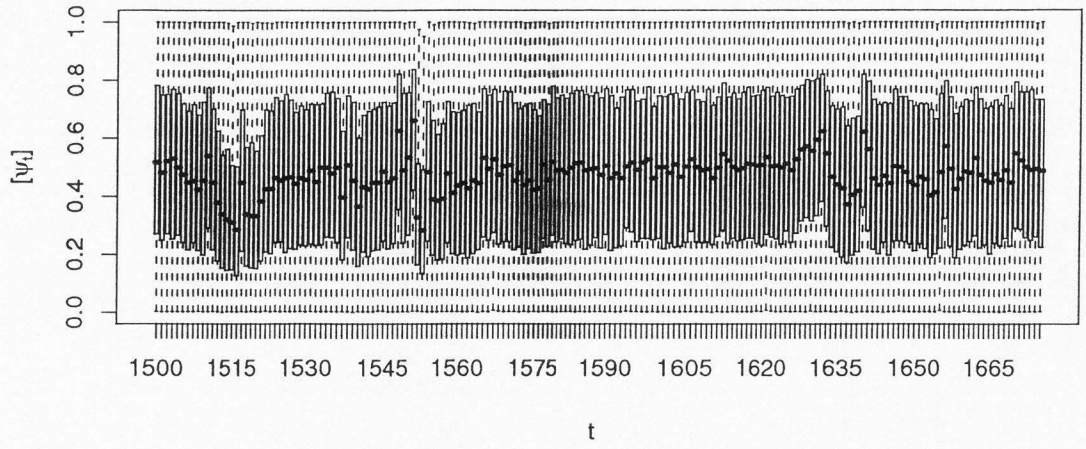

Figure 5: Boxplots for the posterior of ψ_t ($[\psi_t|\boldsymbol{y}]$) from the "ψ_t, z_t" model.

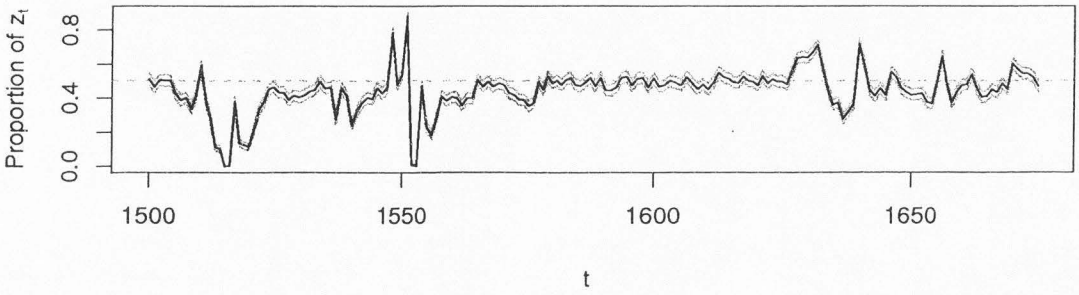

Figure 6: Proportion of z_t with asymptotic confidence intervals from the "ψ_t, z_t" model.

Table 1: Bounds of truncation, posterior mean, and posterior 95% Credible Interval for HYMOD parameters

Parameter	**Truncation Bounds**	**Posterior Mean**	**Posterior 95% Credible Interval**
cmax	[0, 400]	268.4	(243.8, 293.1)
bexp	[0, 2]	0.7431	(0.6409, 0.8593)
alpha	[0, 1]	0.8558	(0.8388, 0.8733)
Rs	[0, 1]	0.004468	(0.003053, 0.006039)
Rq	[0, 1]	0.4757	(0.4633, 0.4889)

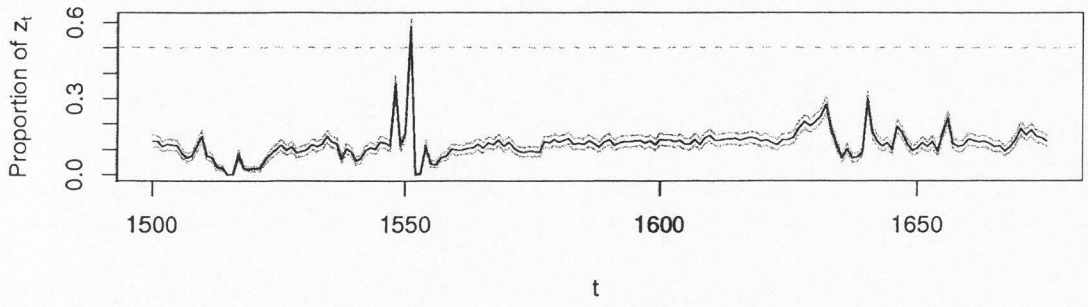

Figure 7: Proportion of z_t with asymptotic confidence intervals from the "ψ, z_t" model.

Table 2: Bounds of truncation, posterior mean, and posterior 95% Credible Interval for SAC-SMA parameters

Parameter	**Truncation Bounds**	**Posterior Mean**	**Posterior 95% Credible Interval**
UZTWM	[1, 150]	23.5	(18.49, 30.07)
UZFWM	[1, 150]	42.7	(39.48, 47.13)
UZK	[0.1, 0.5]	0.3974	(0.1901, 0.495)
PCTIM	[0, 0.1]	0.001163	(0.00003376, 0.004144)
ZPERC	[1, 250]	227.2	(179.8, 249.4)
REXP	[1, 5]	1.038	(1.001, 1.129)
LZTWM	[1, 500]	255.4	(230.3, 287)
LZFSM	[1, 1000]	32.32	(24.17, 40.14)
LZFPM	[1, 1000]	81.26	(71.4, 90.13)
LZSK	[0.01, 0.25]	0.06325	(0.04736, 0.08019)
LZPK	[0.0001, 0.025]	0.004138	(0.002944, 0.005349)
PFREE	[0, 0.6]	0.2779	(0.2518, 0.3152)

Table 3: Deviance information criteria for HYMOD and SAC-SMA data assimilation models and for the "ψ, z" , "ψ, z_t", and "ψ_t, z_t" mixture models.

Model	**DIC**
HYMOD	16274.02
SAC-SMA	16060.42
ψ, z	16060.42
ψ, z_t	16002.55
ψ_t, z_t	15975.23

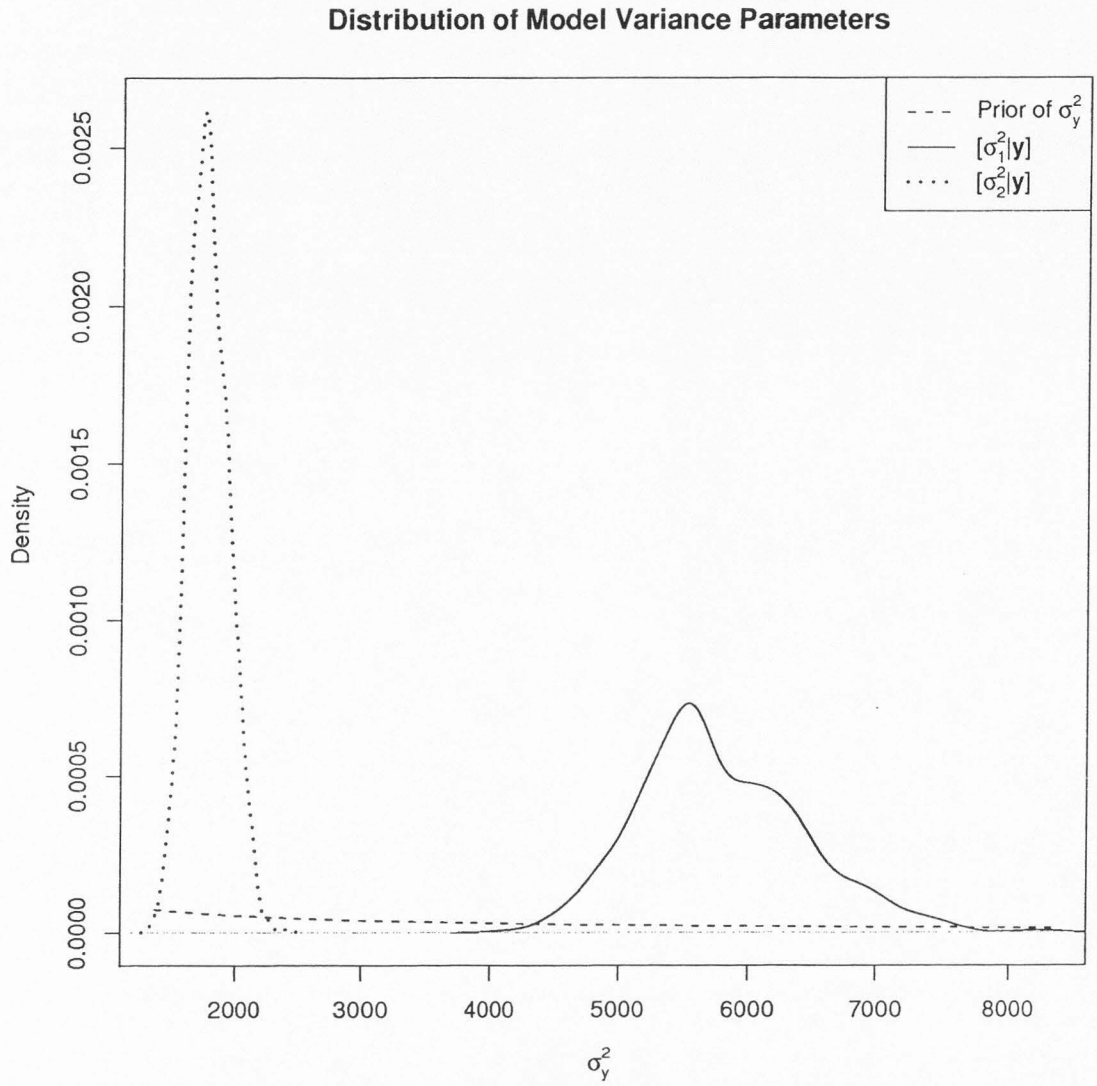

Figure 8: Posterior distributions of HYMOD ($[\sigma_1^2|\boldsymbol{y}]$) and SAC-SMA variance parameters ($[\sigma_1^2|\boldsymbol{y}]$) and corresponding prior distribution.

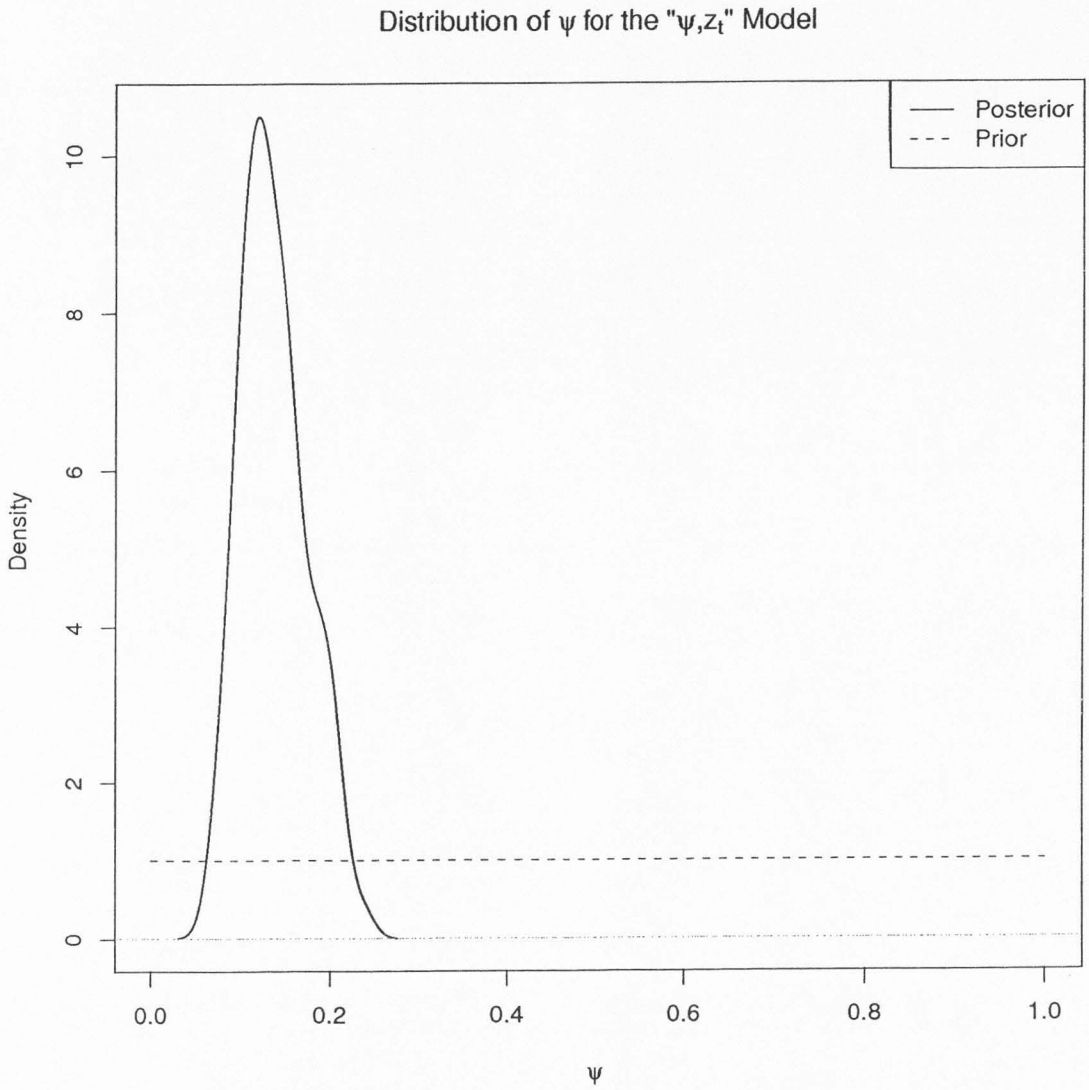

Figure 9: Posterior distribution of ψ ($[\psi|\boldsymbol{y}]$) and corresponding prior distribution from the "ψ, z_t" model.

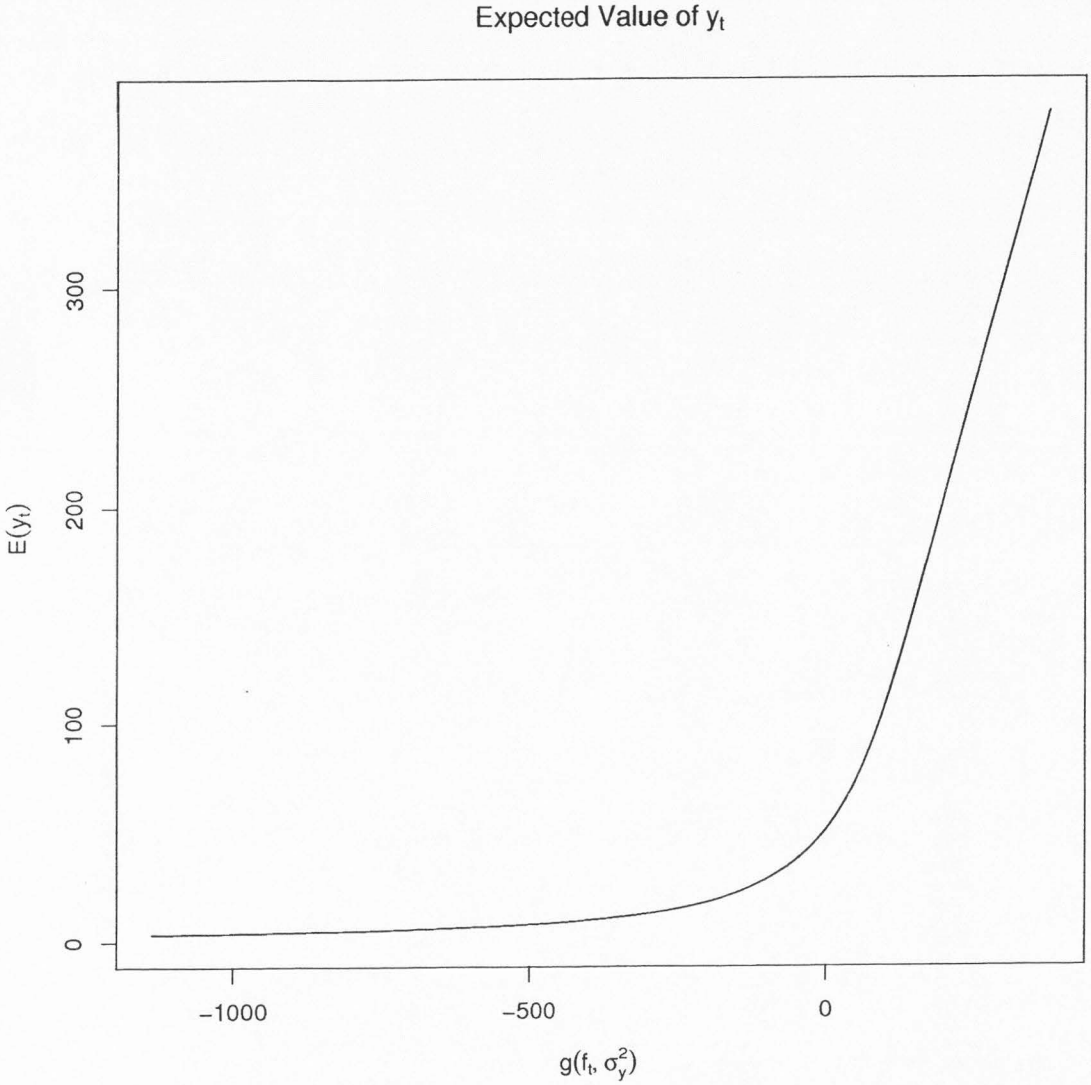

Figure 10: Plot of the expected value of y_t.